Homework 9: Software Design Considerations

Team Code Name: ___2D-MPR_____ Group No. _12_

Team Member Completing This Homework: _Alex Bridge_____

E-mail Address of Team Member: _abridge_____ @ purdue.edu

Evaluation:

SCORE	DESCRIPTION
10	***Excellent*** *– among the best papers submitted for this assignment. Very few corrections needed for version submitted in Final Report.*
9	***Very good*** *– all requirements aptly met. Minor additions/corrections needed for version submitted in Final Report.*
8	***Good*** *– all requirements considered and addressed. Several noteworthy additions/corrections needed for version submitted in Final Report.*
7	***Average*** *– all requirements basically met, but some revisions in content should be made for the version submitted in the Final Report.*
6	***Marginal*** *– all requirements met at a nominal level. Significant revisions in content should be made for the version submitted in the Final Report.*
*	***Below the passing threshold*** *– major revisions required to meet report requirements at a nominal level.* ***Revise and resubmit.***

***** *Resubmissions are due within* ***one week*** *of the date of return, and will be awarded a score of "6" provided all report requirements have been met at a nominal level.*

Comments:

1.0 Introduction

The 2D-MPR will be a waypoint seeking robot with capabilities to autonomously gather depth data from a Kinect USB device in a 360 degree pattern around itself. It will then be able to combine this data from multiple surveys into a working two-dimensional map of the area in which it is. Significant software blocks include drivers for the following devices: LCD screen, Kinect, Xbee, and iRobot Create. The LCD will communicate through an SPI port. The Xbee will use a UART port. The Kinect will be connected through USB, and the iRobot Create will be connected to an array of GPIO pins along with a UART.

2.0 Software Design Considerations

Memory Map

The top-level view of the memory in the LPC1769 is as follows. The flash memory (on-chip non-volatile memory) is located from memory address **0x0000 0000** to memory address **0x0007 FFFF**. The on-chip generic SRAM is then located from **0x1000 0000** to **0x1000 7FFF.** There is a small (8kB) boot ROM on this chip, and it is located from **0x1FFF 0000** to **0x1FFF 1FFF**. There is another block of SRAM from **0x2007 C000** to **0x2009 FFFF**, which is used for holding AHB peripheral data (**0x2007 C000 – 0x2008 3FFF**) and for GPIO (**0x2009 C000 – 0x2009 FFFF**). The addresses for the other peripherals (AHB and APB) come after the GPIO, in the range of **0x4000 0000** to **0x501F FFFF**. Beyond this, there are some private peripherals, described in the documentation as “Cortex-M3 related functions”. These are located between **0xE000 0000** and **0xE00F FFFF**. All of the above is documented in Appendix C, which is excerpted from the *LPC17xx User Manual*.

During boot, the boot ROM takes care of certain initializations and then passes the program counter to user code, which it searches for, making the exact location of the data, variables, stack, heap, and code at the whim of the assembler. This abstracts the construction of the sections of information to a procedure of assembler directives, which are provided by ARM. [2] The startup code is provided in the link. The only restrictions we must make are to keep the

code in the flash memory for persistence sake (which is the default for the assembler), and that we allow enough room to store the point clouds as they come in off the Kinect.

Peripherals

The peripherals all have been assigned specific ports to inhabit, and these assignments are as follows. The UART0 port (**0x4000 C000**) is used for In System Programming, which is the way we have designed the system to be programmed. For control of the iRobot Create, digital outputs are mapped to GPIO port 1 and inputs are mapped to GPIO port 0. Communication with the Xbee wireless transceiver is done through UART1 (**0x4001 0000**). The Kinect communicates through USB to a host onboard the LPC1769. The USB register is located at **0x5000 C000**. The final peripheral that is needed is a debug LCD. This will be interfaced through UART2 (**0x4009 8000**).

In order to enable the pins for the correct functionality, the PINSEL registers must be modified. Two bits represent each pin, and allow for primary and three alternate functionalities. Primary functionality is GPIO, and is enabled by setting the corresponding bits to 00. Therefore to use P0 and P1 for the iRobot Create, we would clear PINSEL0-3, which correspond to P0 and P1. The configuration for the UART's and the USB are the first alternate function, so the PINSEL for their respective pins would be set to 01. In addition to the PINSEL register, there is a PINMODE register for setting whether the respective pin uses an internal pull-up resistor, pull-down resistor, or neither. This is all the configuration needed for GPIO. For USB, more care must be taken. In the PCONP register, the PCUSB bit must be set in order to enable USB functionality. The USB clock must be configured, which is located at PLL1. This must be configured to 48MHz. For the UART ports, configuration is slightly different. In the PCONP register, the bits corresponding to UART0-2 must be set high. In the PCLKSEL registers, the PCLK_UARTx for the correct UART's functioning must be selected. Then the baud rate may be configured by enabling the DLAB bit in the UxLCR register, and then changing the DLL and DLM registers to achieve the correct baud rate. This will correctly configure the UART's for Tx/Rx.

Code Organization

The application code will be primarily flag driven, with interrupts enabled for obstacle detection. Interrupts may need to be enabled for the USB, however, all other processing will be blocked during the Kinect survey taking, so said interrupts would be only to enable the USB stack to perform correctly. The code will be purely procedural during a survey, not accepting any commands from the Xbee during that time. Once a survey is taken and the updated map is sent to the host PC, the software will begin to monitor the flag for the communication channel in order to receive its next set of waypoints. The flag will continue to be monitored during transit to the next survey location, allowing for in-path modification of the travel path. Interrupts will be enabled for the bumper and cliff sensors on the iRobot Create, so that immediate action can be taken in case of obstacle. Once the robot assumes its new survey position, the flag will no longer be monitored and the interrupts will be disabled.

The top level design of the software is relatively simple, due to the single-task tendencies of the project. It is impossible to select a new survey location without having computed an updated map, and it is impractical to stop the robot mid-survey to go to a new location. This leads to a simple flag-driven system with a single interrupt for the time-critical portion of the project. The interrupts/flags are disabled during surveys in order to allow the survey to be completed once started.

Debugging

We have a few provisions for debugging, including the debug LCD, which will display status updates wherever they are enabled. We also have a heartbeat LED to make sure that the processor is still running. The simplest way to test the functionality is to allow the robot to make one survey where it sits and make sure that the data received at the PC is legitimate.

3.0 Software Design Narrative

There will be a main loop that executes at the beginning of the user code. This loop will have control over the drivers for the Xbee, the Kinect, the LCD, and the iRobot. From the loop, a function will be called when not in position to take a survey that navigates the robot to the next

survey coordinates. When in position, another function will be called that controls the Kinect and the iRobot to allow a survey to be taken. During this time interrupts will be disabled. Once the function processes the survey data, it will hand control off to the map processing function, which will generate best-fit lines using a K-means algorithm, which will hand control to a map stitching function, which will fit the current map with the cumulative map, likely using the K-means algorithm again. Once this is complete, the updated map will be sent through the Xbee driver to be sent to the PC. Then, control will be restored to the main loop to wait for a response from the PC.

The Kinect driver will wrap around the USB driver, which allows coherent USB communication to a device. The Kinect driver will be responsible for sending commands to the Kinect (set_tilt, get_depth, etc.) and to receive any data that comes back from the Kinect. It will also filter out unnecessary data, pruning the depth images to the 4x79 sliver that will be used as the point cloud. We have open-source drivers for the Kinect, but they are only for the PC at the moment. We are currently working on porting those drivers over to the microcontroller.

The Xbee driver will format and send data to the Xbee unit when required, and it will receive waypoint data and store it in an appropriate queue. The protocol for this communication has been established, but software has only been written for the PC side at this point.

The LCD driver acts as a simple way to send messages to the LCD. This driver has not been written yet.

The iRobot driver will send commands to the robot to move it to where it is needed. It will also receive data for detecting obstacles. This means that the input pins from the iRobot will need to be enabled for interruptability. This driver has not been written yet.

The survey coherence algorithm will take the data points from the Kinect and adjust for the current position and orientation of the robot. This is prior to line fitting, so that the created lines are in the same coordinate system as the existing map. This program will be a simple trigonometric function on each data point depending on the current location of the robot (by dead reckoning). This algorithm has not yet been started.

The line fitting algorithm will take the coherent point cloud from the survey coherence algorithm and will fit lines to the points using K-means. This will then be ready for integration with the cumulative map. This algorithm has been written and tested on the PC, and is ready to be ported to the microcontroller.

The map stitching algorithm will take the generated lines from the line fitting algorithm and fit them onto the existing map, adding any new data that is included. This will likely use the similar K-means algorithm, with corners replacing the points in the line-fitting algorithm. This algorithm has not yet been started.

4.0 Summary

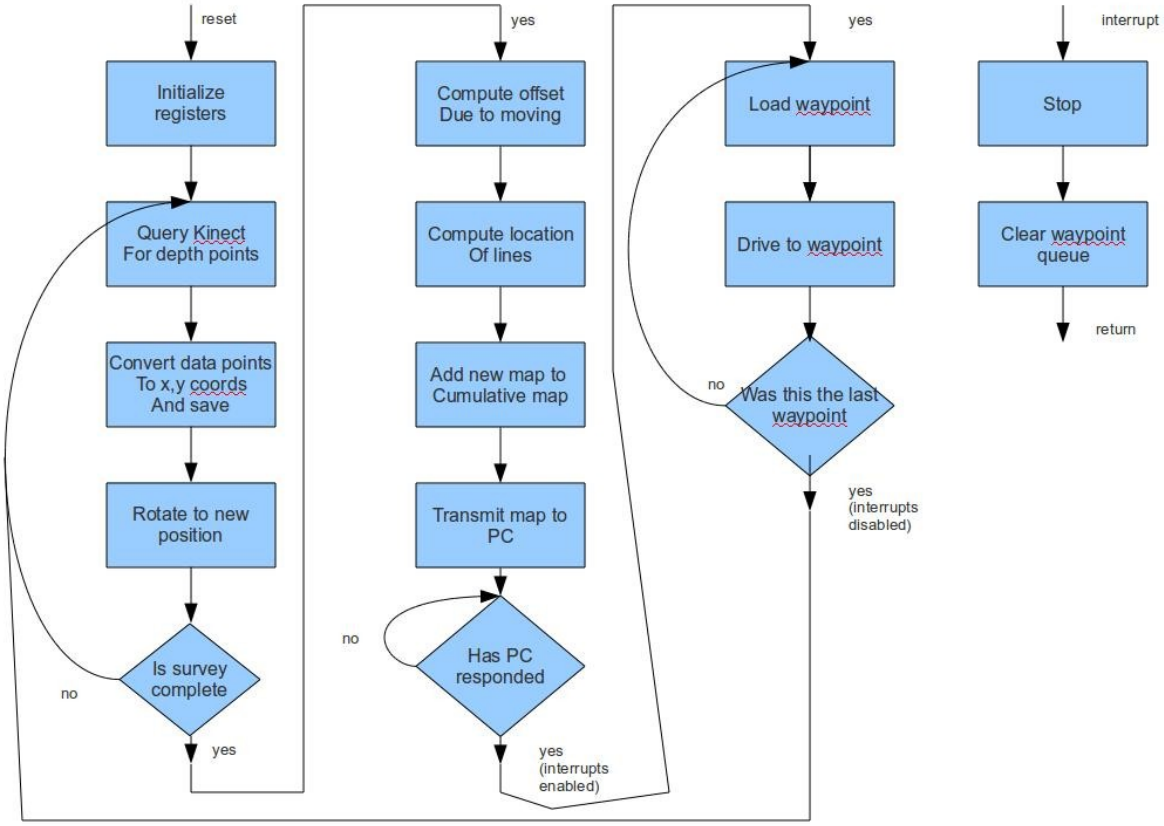
The software for this project is primarily based around drivers for the various peripherals, with a small burst of computation after each survey that is taken. The peripherals will generally be used one at a time, reducing the need for interrupt driven architecture, except when there is a chance of cataclysm (when the robot is moving). The code takes on a very formalized pattern of order, following the chain: “survey”, “compute”, “transmit”, “wait”, “receive”, “go while listening for interrupt”, “stop”, “survey”. The primary difficulties of the software from this point forward will be porting the drivers and algorithms to the microcontroller, as the most complex of those drivers and algorithms have already been proven on the PC.

List of References

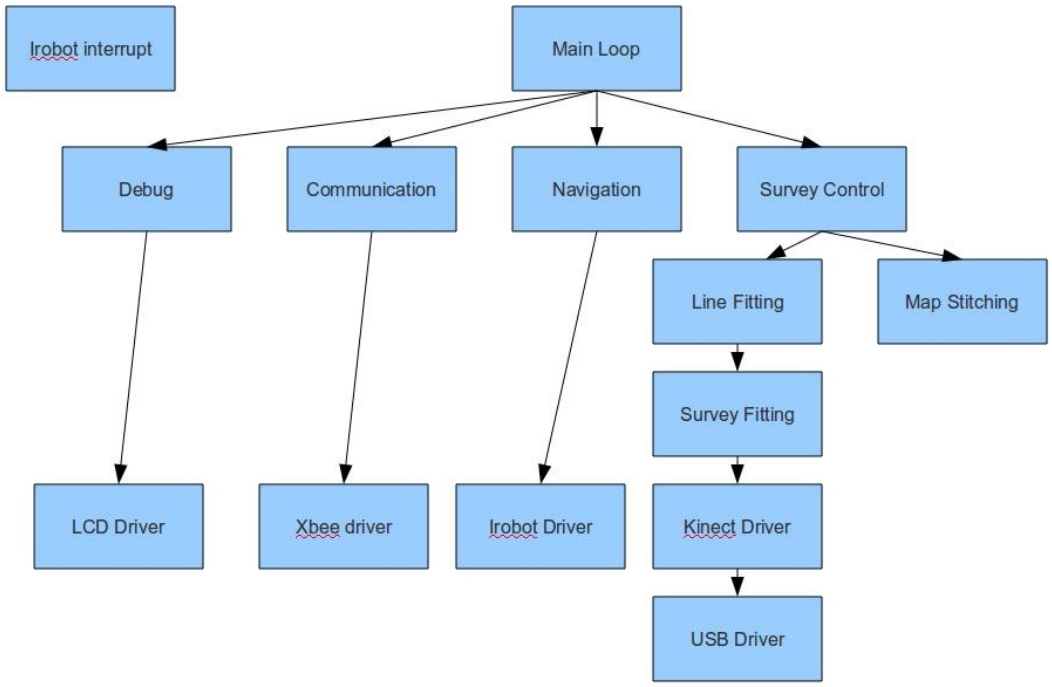
[1] NXP Semiconductors, *LPC 1769 User Manual*,
www.nxp.com/documents/**user_manual**/UM10360.pdf
[Accessed: Mar. 25, 2011].

[2] ARM Ltd, *CMSIS Overview,*
http://www.keil.com/support/man/docs/gsac/gsac_cmsisoverview.htm
[Accessed: Mar. 25, 2011].

Appendix A: Flowchart/Pseudo-code for Main Program



Appendix B: Hierarchical Block Diagram of Code Organization



Appendix C: Top-Level Memory Map

Table 3. LPC17xx memory usage and details

Address range	General Use	Address range details and description	
0x0000 0000 to 0x1FFF FFFF	On-chip non-volatile memory	0x0000 0000 - 0x0007 FFFF	For devices with 512 kB of flash memory.
		0x0000 0000 - 0x0003 FFFF	For devices with 256 kB of flash memory.
		0x0000 0000 - 0x0001 FFFF	For devices with 128 kB of flash memory.
		0x0000 0000 - 0x0000 FFFF	For devices with 64 kB of flash memory.
		0x0000 0000 - 0x0000 7FFF	For devices with 32 kB of flash memory.
	On-chip SRAM	0x1000 0000 - 0x1000 7FFF	For devices with 32 kB of local SRAM.
		0x1000 0000 - 0x1000 3FFF	For devices with 16 kB of local SRAM.
		0x1000 0000 - 0x1000 1FFF	For devices with 8 kB of local SRAM.
	Boot ROM	0x1FFF 0000 - 0x1FFF 1FFF	8 kB Boot ROM with flash services.
0x2000 0000 to 0x3FFF FFFF	On-chip SRAM (typically used for peripheral data)	0x2007 C000 - 0x2007 FFFF	AHB SRAM - bank 0 (16 kB), present on devices with 32 kB or 64 kB of total SRAM.
		0x2008 0000 - 0x2008 3FFF	AHB SRAM - bank 1 (16 kB), present on devices with 64 kB of total SRAM.
	GPIO	0x2009 C000 - 0x2009 FFFF	GPIO.
0x4000 0000 to 0x5FFF FFFF	APB Peripherals	0x4000 0000 - 0x4007 FFFF	APB0 Peripherals, up to 32 peripheral blocks, 16 kB each.
		0x4008 0000 - 0x400F FFFF	APB1 Peripherals, up to 32 peripheral blocks, 16 kB each.
	AHB peripherals	0x5000 0000 - 0x501F FFFF	DMA Controller, Ethernet interface, and USB interface.
0xE000 0000 to 0xE00F FFFF	Cortex-M3 Private Peripheral Bus	0xE000 0000 - 0xE00F FFFF	Cortex-M3 related functions, includes the NVIC and System Tick Timer.